

Шаблон отчёта по лабораторной работе

Пакавира Арсениу

Лаборатория № 8

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация циклов в NASM.	9
4.2	Обработка аргументов командной строки.	14
4.3	Задание для самостоятельной работы.	19
5	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	создание файлов.....	Erro! Indicador não definido.
4.2	ввод текста	8
4.3	запуск исполняемого файла	9
4.4	изменение текста программы	10
4.5	запуск обновленной файла	11
4.6	изменение текста программы	11
4.7	запуск исполняемого файла	12
4.8	ввод текста	13
4.9	запуск исполняемого файла	13
4.10	ввод текста	14
4.11	запуск исполняемого файла	15
4.12	изменение текста программы	16
4.13	запуск исполняемого файла	17
4.14	текст программы	18
4.15	запуск исполняемого файла	18

Списоктаблиц

1 Цель работы

- Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

- Реализация циклов в NASM.
- Обработка аргументов командной строки.
- Задание для самостоятельной работы.

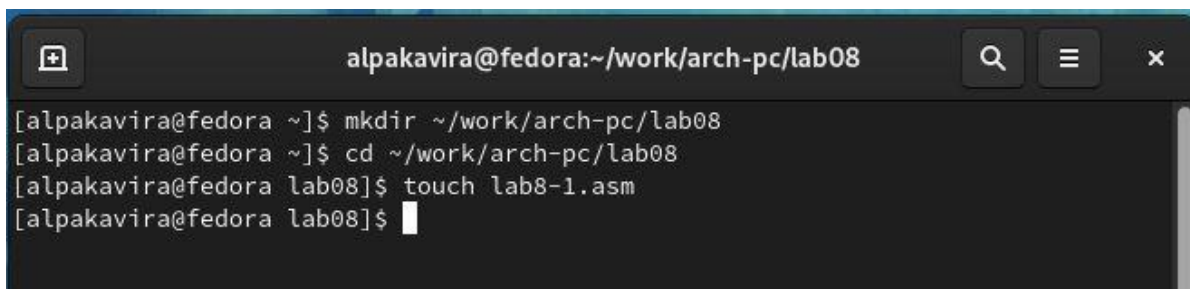
3 Теоретическое введение

- Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается.
- Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек.
- Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как «мусор», который будет перезаписан при записи нового значения в стек.
- Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM.

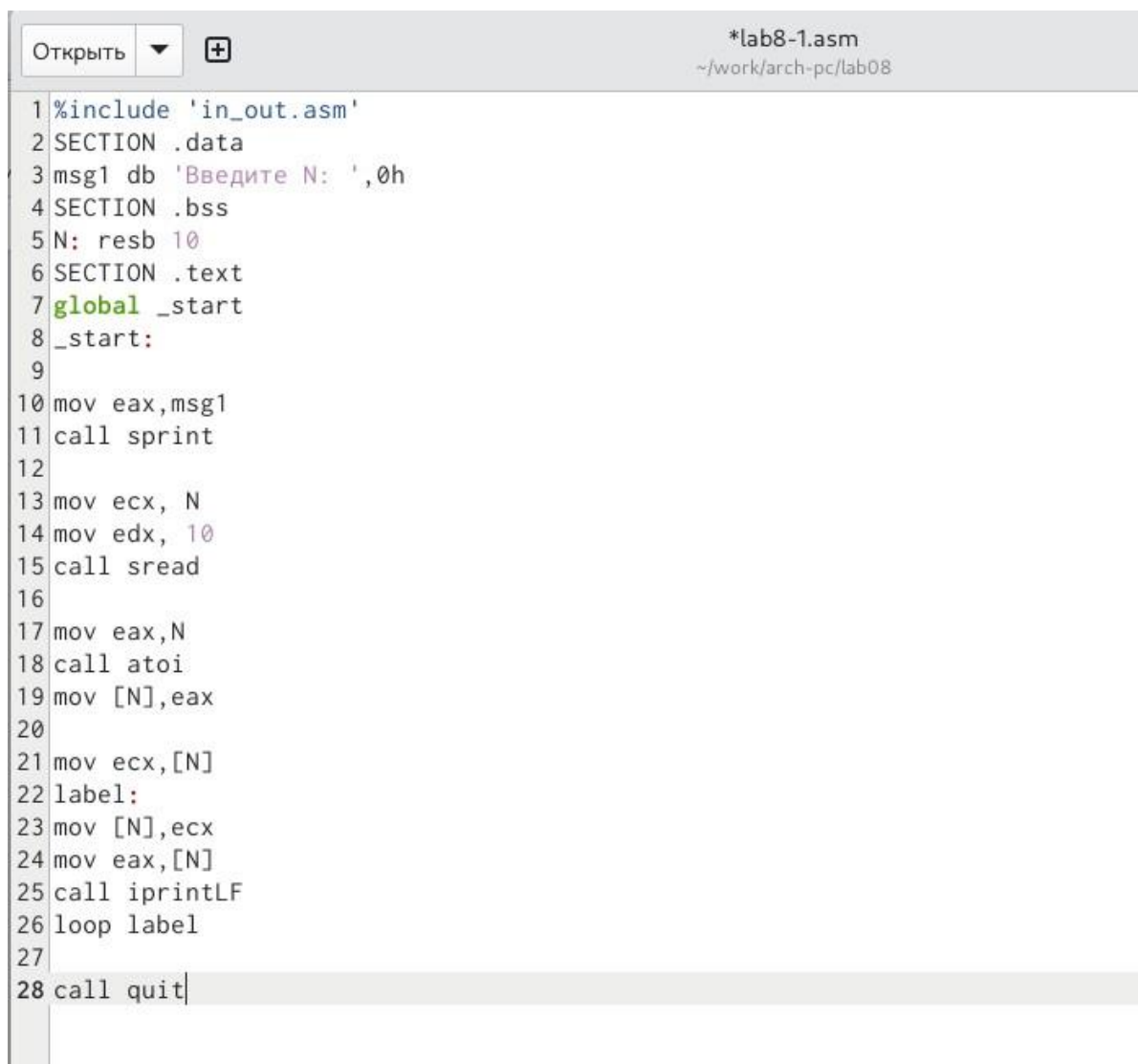
- Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm.(рис.[4.1]).



```
alpakavira@fedora:~/work/arch-pc/lab08
[alpakavira@fedora ~]$ mkdir ~/work/arch-pc/lab08
[alpakavira@fedora ~]$ cd ~/work/arch-pc/lab08
[alpakavira@fedora lab08]$ touch lab8-1.asm
[alpakavira@fedora lab08]$
```

Рис. 4.1: создание файлов

- Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис.[4.2]).



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9
10 mov eax,msg1
11 call sprint
12
13 mov ecx, N
14 mov edx, 10
15 call sread
16
17 mov eax,N
18 call atoi
19 mov [N],eax
20
21 mov ecx,[N]
22 label:
23 mov [N],ecx
24 mov eax,[N]
25 call iprintLF
26 loop label
27
28 call quit
```

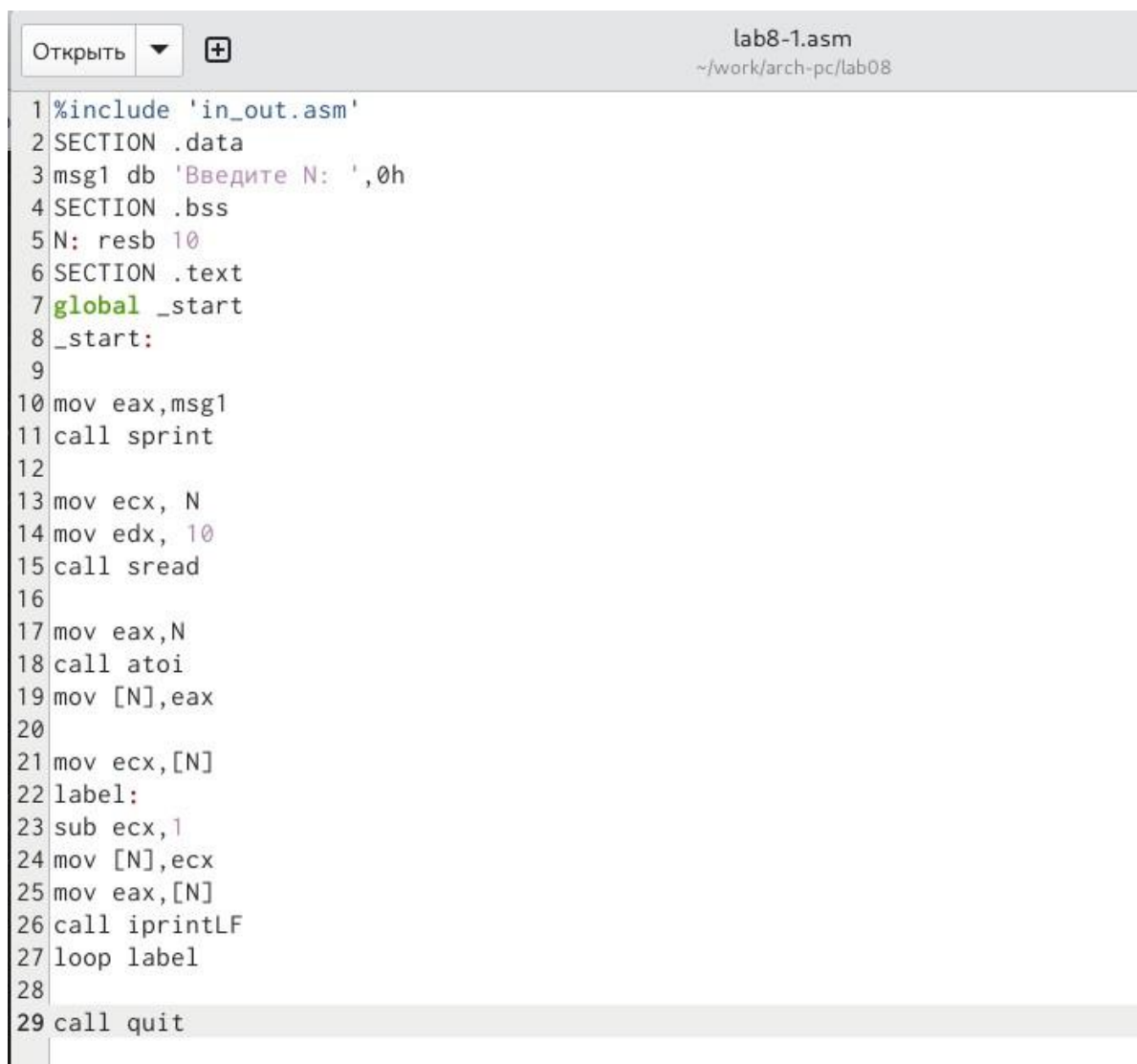
Рис. 4.2: ввод текста

- Создаю исполняемый файл и проверяю его работу. (рис.[4.3]).


```
[alpakavira@fedora lab08]$ nasm -f elf lab8-1.asm
[alpakavira@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[alpakavira@fedora lab08]$ ./lab8-1
Введите N: 6
6
5
4
3
2
1
[alpakavira@fedora lab08]$
```

Рис. 4.3: запуск исполняемого файла

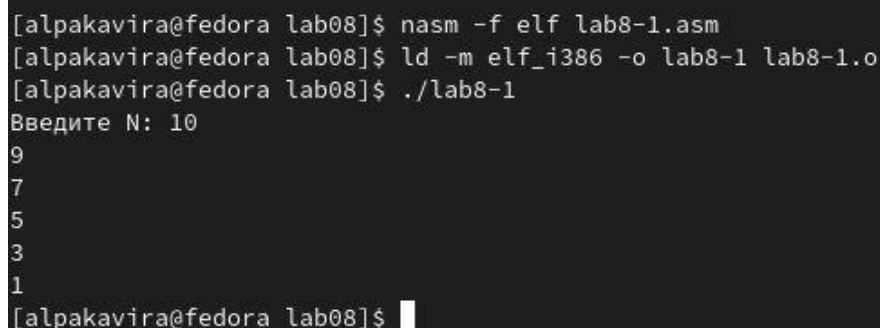
- Изменяю текст программы, добавив изменение значения регистра есх в цикле.
(рис.[4.4]).



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9
10 mov eax,msg1
11 call sprint
12
13 mov ecx, N
14 mov edx, 10
15 call sread
16
17 mov eax,N
18 call atoi
19 mov [N],eax
20
21 mov ecx,[N]
22 label:
23 sub ecx,1
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF
27 loop label
28
29 call quit
```

Рис. 4.4: изменение текста программы

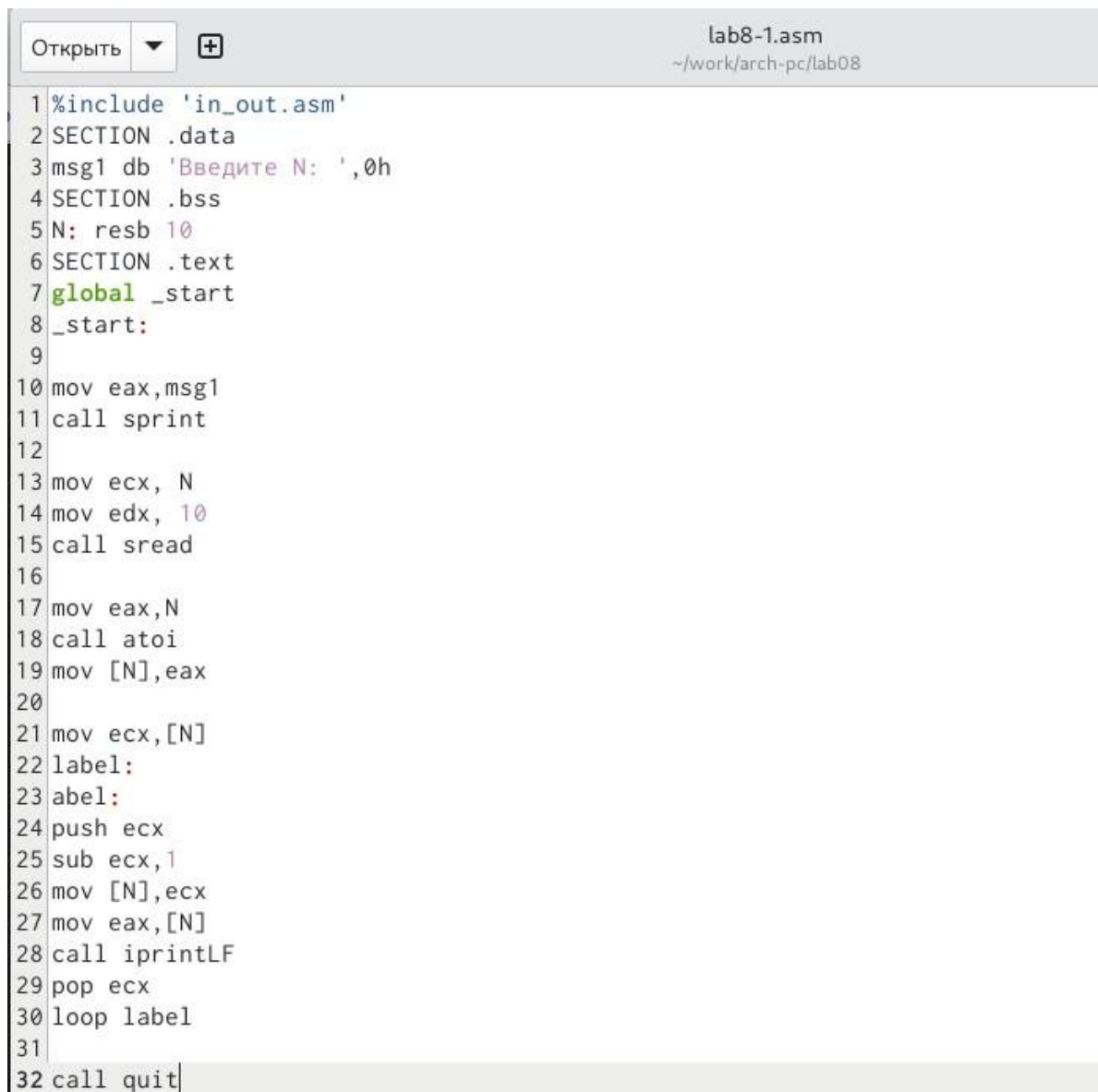
- Создаю исполняемый файл и проверяю его работу. (рис.[4.5]).



```
[alpakavira@fedora lab08]$ nasm -f elf lab8-1.asm
[alpakavira@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[alpakavira@fedora lab08]$ ./lab8-1
Введите N: 10
9
7
5
3
1
[alpakavira@fedora lab08]$
```

Рис. 4.5: запуск обновленной файла

- Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop. (рис. [4.6]).



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9
10 mov eax,msg1
11 call sprint
12
13 mov ecx, N
14 mov edx, 10
15 call sread
16
17 mov eax,N
18 call atoi
19 mov [N],eax
20
21 mov ecx,[N]
22 label:
23 label:
24 push ecx
25 sub ecx,1
26 mov [N],ecx
27 mov eax,[N]
28 call iprintLF
29 pop ecx
30 loop label
31
32 call quit
```

Рис. 4.6: изменение текста программы

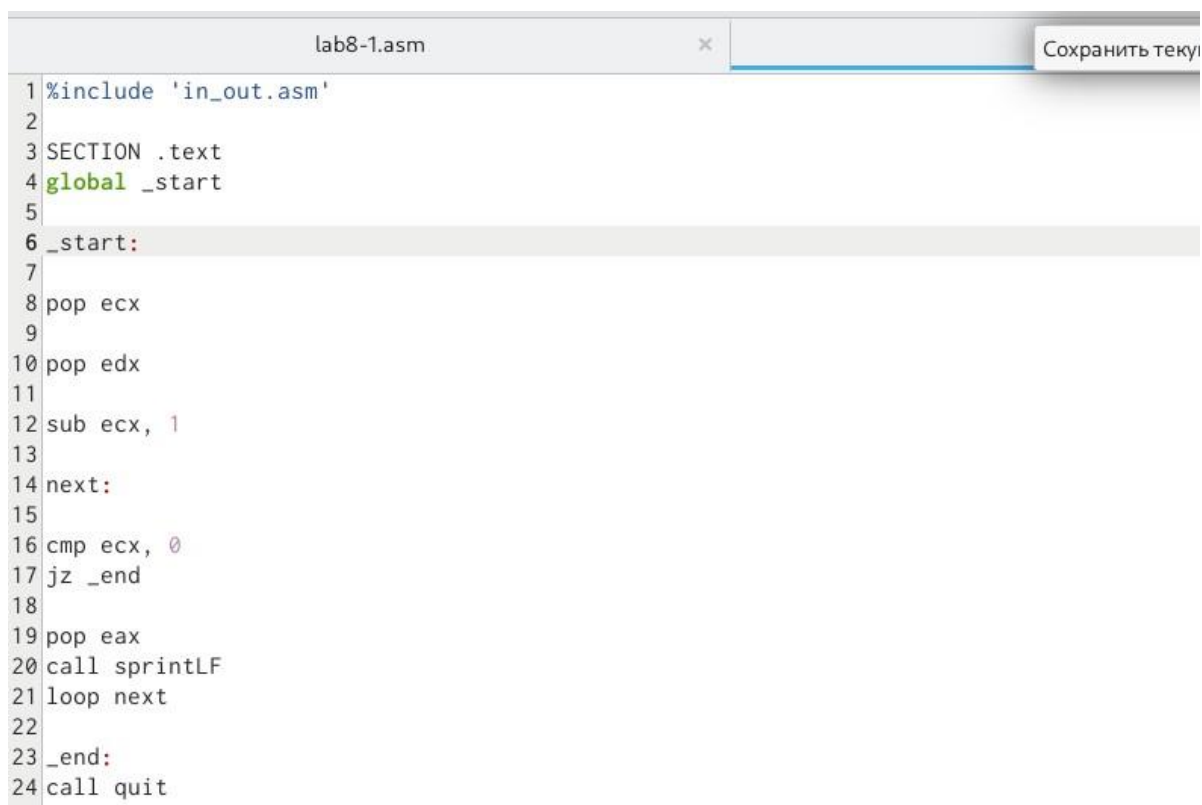
- Создаю исполняемый файл и проверяю его работу. (рис.[4.7]).

```
[alpakavira@fedora lab08]$ nasm -f elf lab8-1.asm
[alpakavira@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[alpakavira@fedora lab08]$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
[alpakavira@fedora lab08]$
```

Рис. 4.7: запуск исполняемого файла

4.2 Обработка аргументов командной строки.

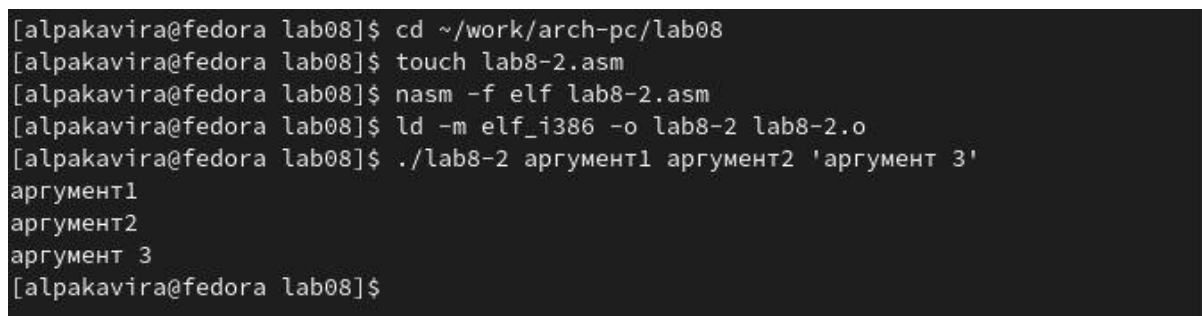
- На этом шаге мы создали файл lab8-2.asm, затем заполнили в нем наш код. (рис.[4.8]).



```
lab8-1.asm
1 %include 'in_out.asm'
2
3 SECTION .text
4 global _start
5
6 _start:
7
8 pop ecx
9
10 pop edx
11
12 sub ecx, 1
13
14 next:
15
16 cmp ecx, 0
17 jz _end
18
19 pop eax
20 call sprintLF
21 loop next
22
23 _end:
24 call quit
```

Рис. 4.8: ввод текста

- Создаю исполняемый файл и запускаю его, указав нужные аргументы.
(рис.[4.9]).

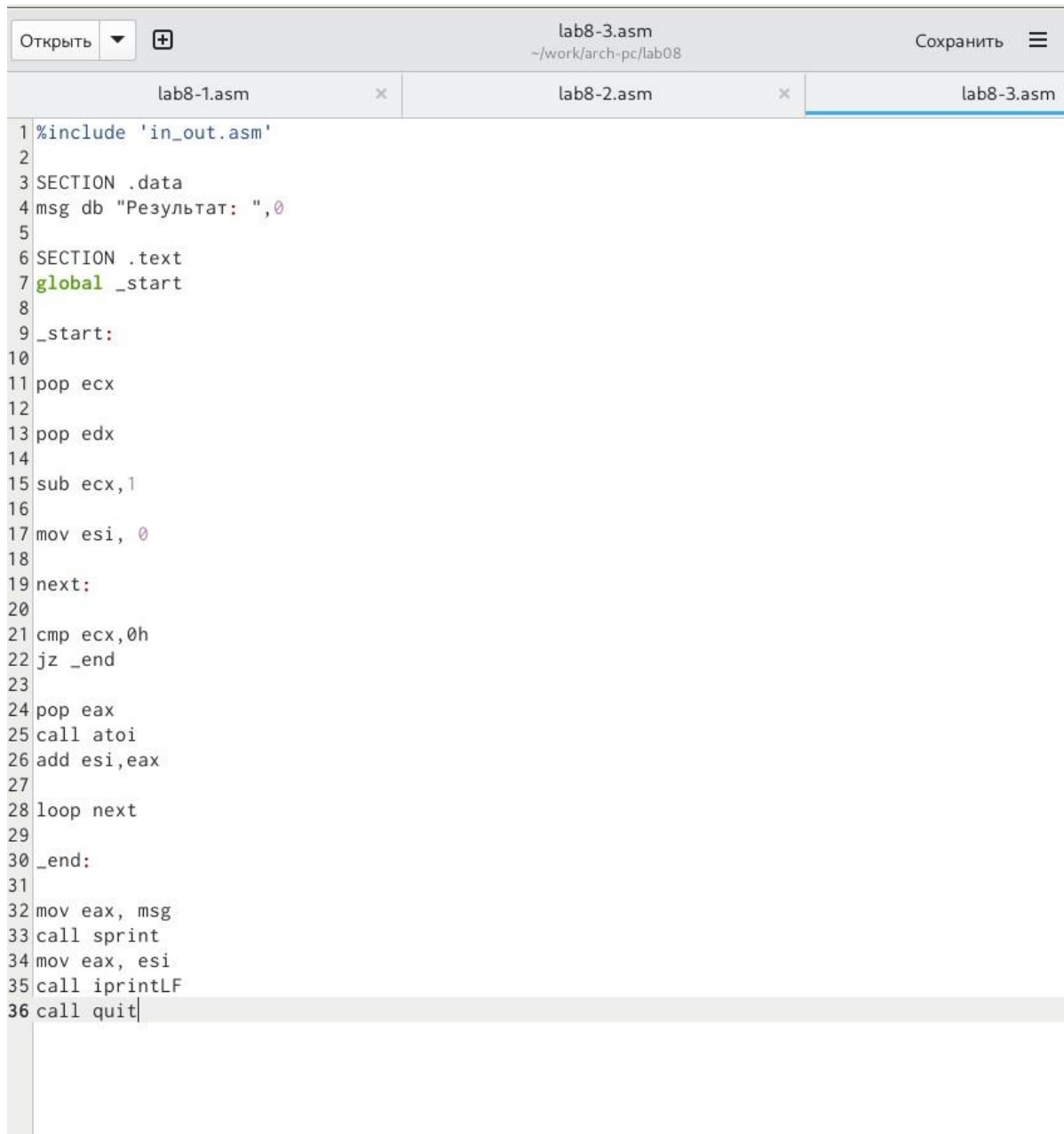


```
[alpakavira@fedora lab08]$ cd ~/work/arch-pc/lab08
[alpakavira@fedora lab08]$ touch lab8-2.asm
[alpakavira@fedora lab08]$ nasm -f elf lab8-2.asm
[alpakavira@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[alpakavira@fedora lab08]$ ./lab8-2 аргумент1 аргумент2 'аргумент 3'
аргумент1
аргумент2
аргумент 3
[alpakavira@fedora lab08]$
```

Рис. 4.9: запуск исполняемого файла

- И, как вы можете видеть, на этот раз при запуске программы мы добавили в команду три аргумента, и в этом случае были обработаны три аргумента

- Первым делом мы создали файл lab8-3.asm, затем заполнили кодом программы. (рис.[4.10]).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8
9 _start:
10
11 pop ecx
12
13 pop edx
14
15 sub ecx,1
16
17 mov esi, 0
18
19 next:
20
21 cmp ecx,0h
22 jz _end
23
24 pop eax
25 call atoi
26 add esi,eax
27
28 loop next
29
30 _end:
31
32 mov eax, msg
33 call sprint
34 mov eax, esi
35 call iprintLF
36 call quit
```

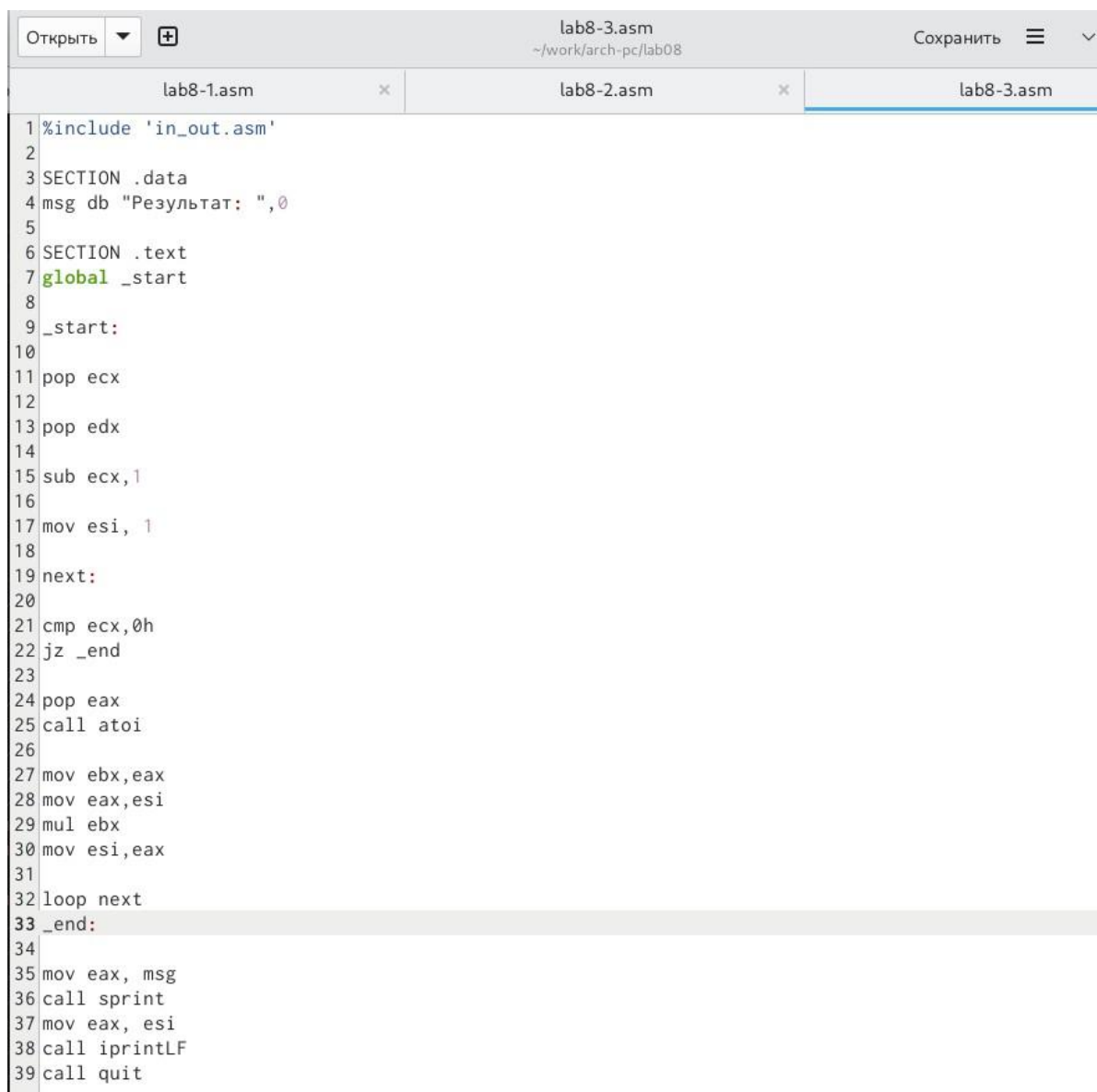
Рис. 4.10: ввод текста

- Создаю исполняемый файл и запускаю его, указав аргументы.(рис.[4.11]).

```
[alpakavira@fedora lab08]$ cd ~/work/arch-pc/lab08
[alpakavira@fedora lab08]$ touch lab8-3.asm
[alpakavira@fedora lab08]$ nasm -f elf lab8-3.asm
[alpakavira@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[alpakavira@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[alpakavira@fedora lab08]$
```

Рис. 4.11: запуск исполняемого файла

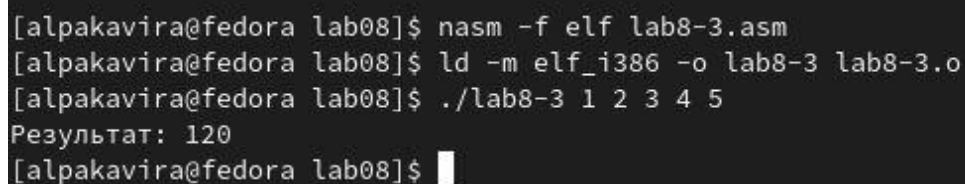
- Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис.[4.12]).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8
9 _start:
10
11 pop ecx
12
13 pop edx
14
15 sub ecx,1
16
17 mov esi, 1
18
19 next:
20
21 cmp ecx,0h
22 jz _end
23
24 pop eax
25 call atoi
26
27 mov ebx,eax
28 mov eax,esi
29 mul ebx
30 mov esi,eax
31
32 loop next
33 _end:
34
35 mov eax, msg
36 call sprint
37 mov eax, esi
38 call iprintLF
39 call quit
```

Рис. 4.12: изменение текста программы

- Создаю исполняемый файл и запускаю его, указав аргументы. (рис.[4.13]).

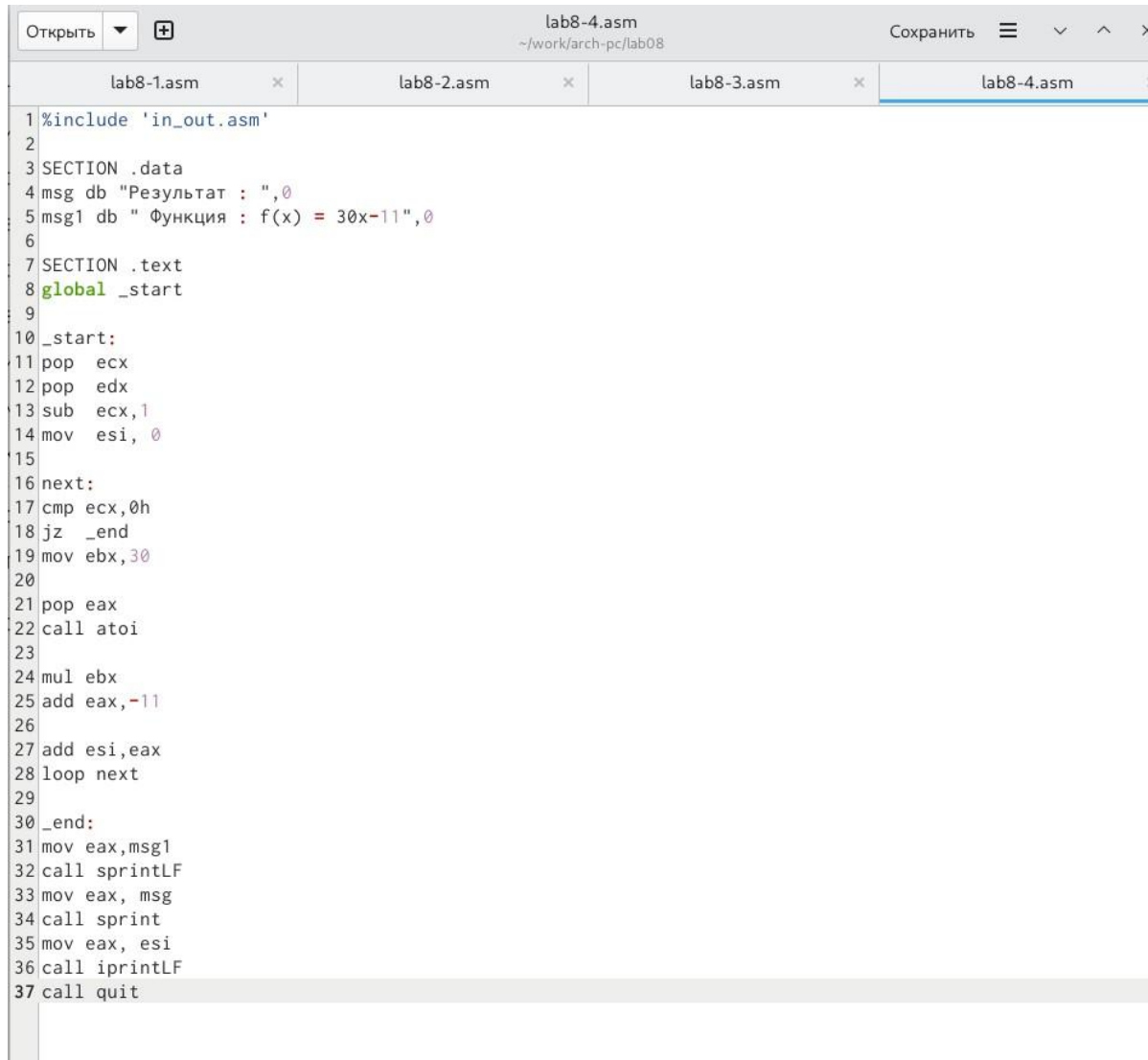


```
[alpakavira@fedora lab08]$ nasm -f elf lab8-3.asm
[alpakavira@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[alpakavira@fedora lab08]$ ./lab8-3 1 2 3 4 5
Результат: 120
[alpakavira@fedora lab08]$
```


Рис. 4.13: запуск исполняемого файла

4.3 Задание для самостоятельной работы.

- В этой части мы должны были написать программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$
- сначала мы создали наш файл lab8-4.asm, где будет находиться наш код, затем мы написали программу. (рис.[4.14]).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат : ",0
5 msg1 db " Функция : f(x) = 30x-11",0
6
7 SECTION .text
8 global _start
9
10 _start:
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 mov ebx,30
20
21 pop eax
22 call atoi
23
24 mul ebx
25 add eax,-11
26
27 add esi,eax
28 loop next
29
30 _end:
31 mov eax,msg1
32 call sprintf
33 mov eax, msg
34 call sprintf
35 mov eax, esi
36 call iprintLF
37 call quit
```

Рис. 4.14: текст программы

- Создаю исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$. (рис.[4.15]).

```
[alpakavira@fedora lab08]$ cd ~/work/arch-pc/lab08
[alpakavira@fedora lab08]$ touch lab8-4.asm
[alpakavira@fedora lab08]$ nasm -f elf lab8-4.asm
[alpakavira@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[alpakavira@fedora lab08]$ ./lab8-4 1 2 3 4
функция :  $f(x) = 30x - 11$ 
результат :
256
[alpakavira@fedora lab08]$
```

Рис. 4.15: запуск исполняемого файла

5 Выводы

- Благодаря этой лабораторной работе мы научились писать программы с использованием циклов и обработки аргументов командной строки, что поможет нам в дальнейшей лабораторной работе.

Список литературы

::: {#refs} :::